CAPE-OPEN Numerical Solver Interfaces

Ben Keeping

Process Systems Enterprise Ltd

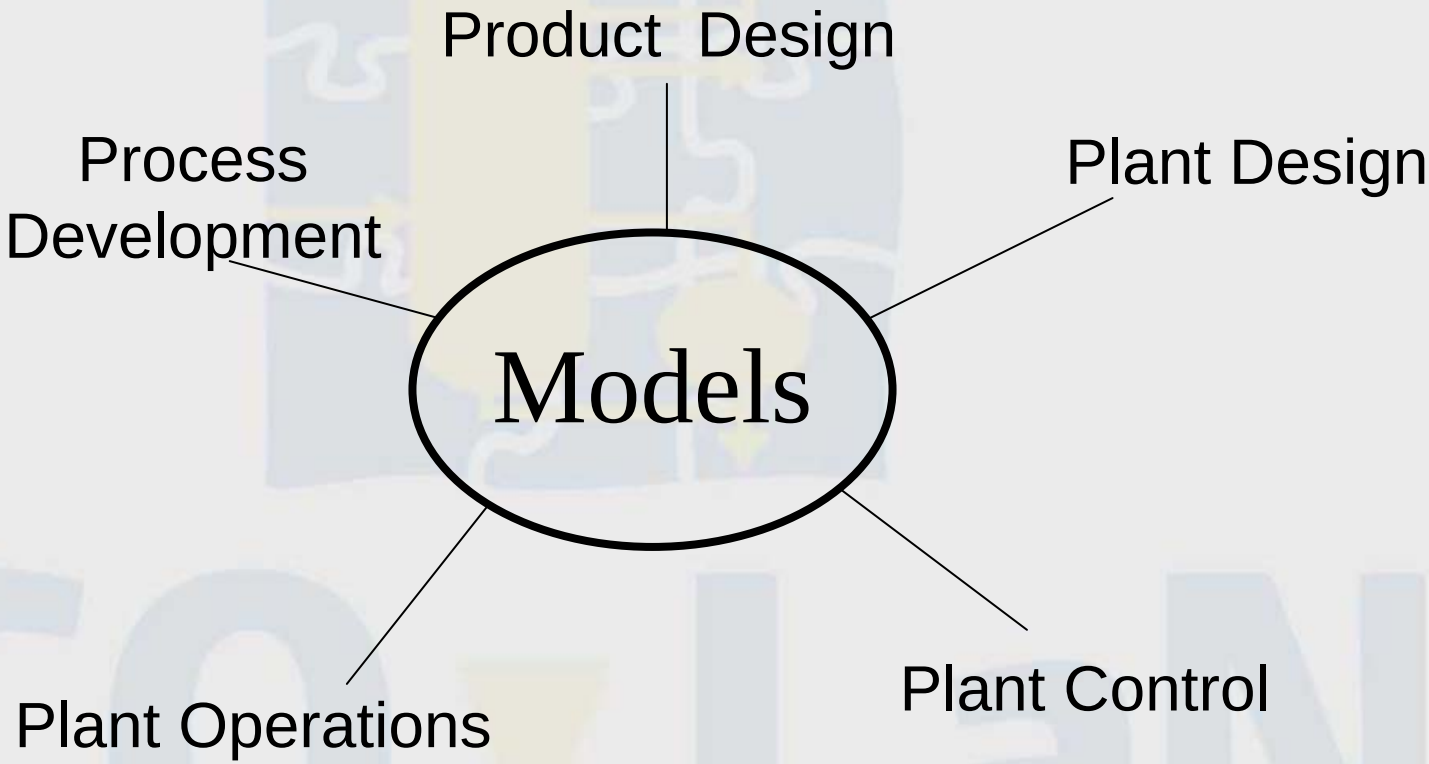
Overview

- Background and scope of CAPE-OPEN Numerical Solvers
- The challenge
- General principles of CAPE-OPEN Numerics interfaces
- Illustration – implementing simple nonlinear solver
- Concluding remarks

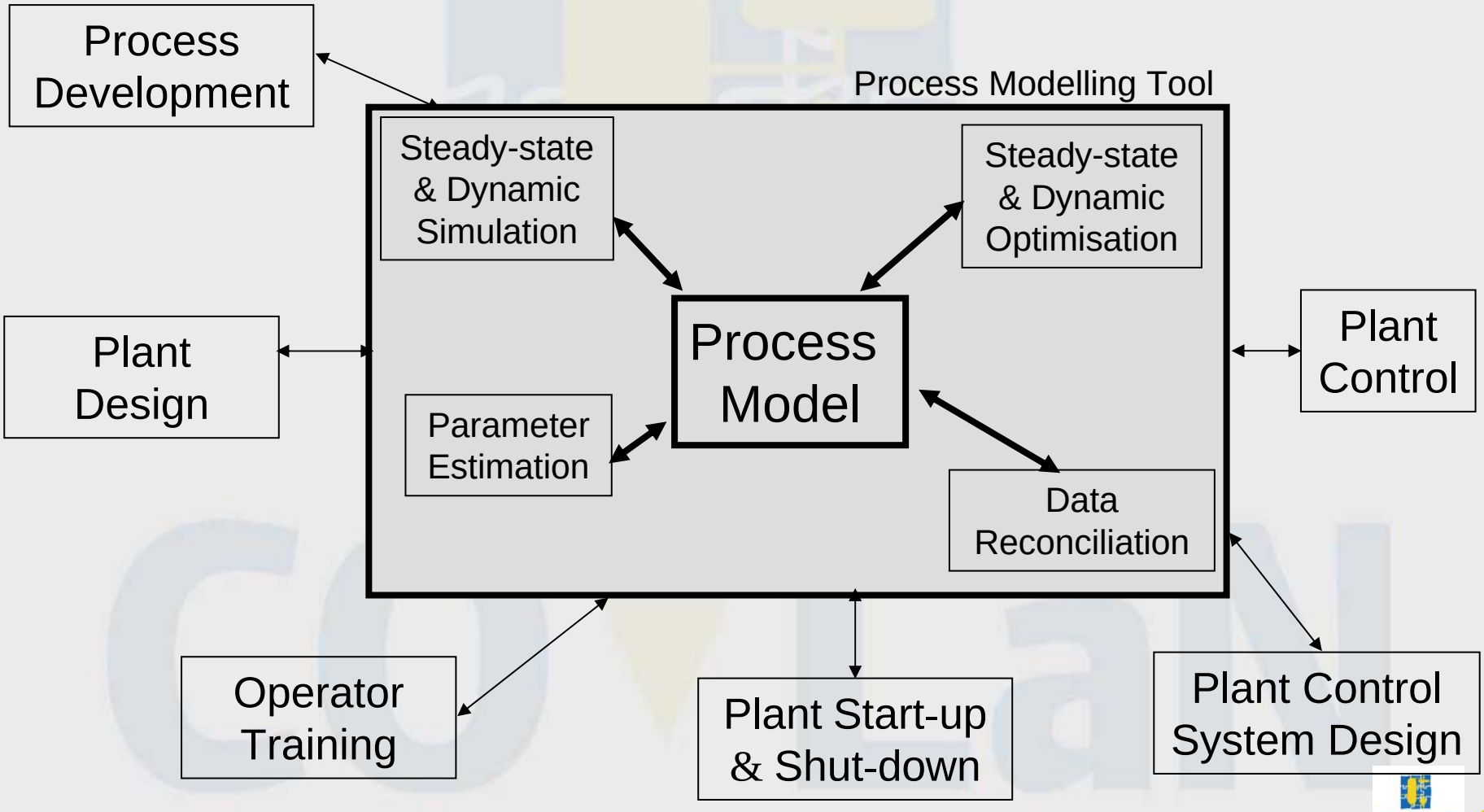
Overview

- **Background and scope of CAPE-OPEN Numerical Solvers**
- The challenge
- General principles of CAPE-OPEN Numerics interfaces
- Illustration – implementing simple nonlinear solver
- Concluding remarks

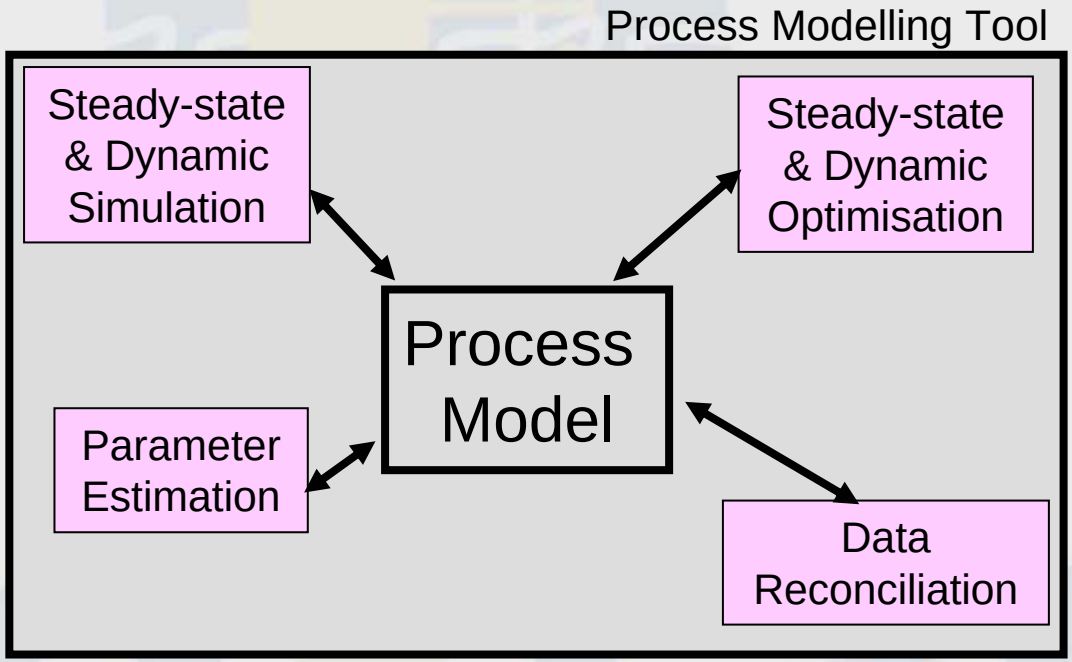
Model-centric process engineering



General-purpose process modelling tools



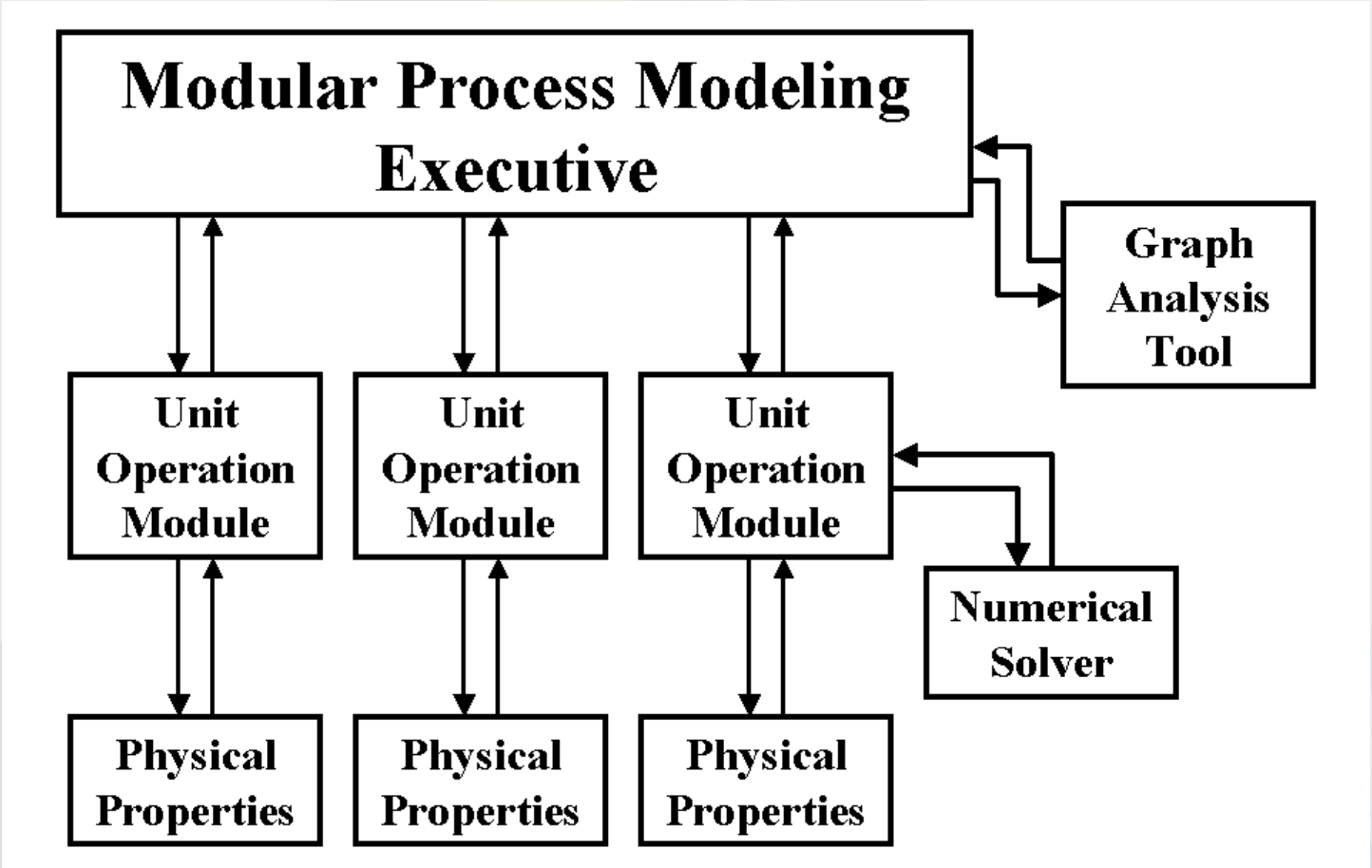
Core model-based activities



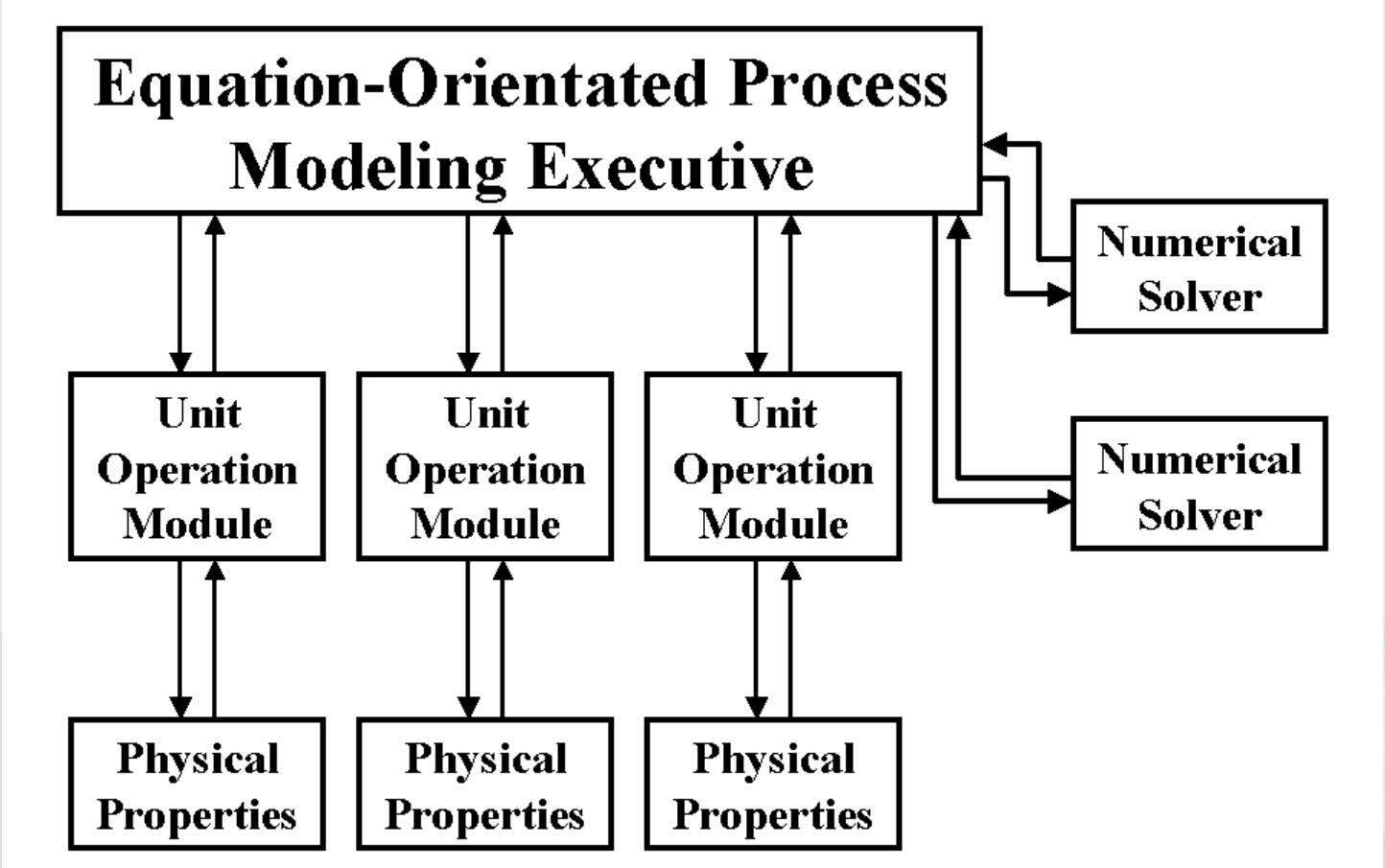
CAPE-OPEN scope for numerical solvers

- Variety of core model-based activities
  - Steady-state & dynamic simulation
  - Steady-state optimisation
  - Parameter estimation and data reconciliation
- Both “modular” and “equation-orientated” systems
- Emphasis on “large-scale” problems

Typical modular process modelling tool



Typical Equation-Orientated process modelling tool



Overview

- Background and scope of CAPE-OPEN Numerical Solvers
- **The challenge**
- General principles of CAPE-OPEN Numerics interfaces
- Illustration – implementing simple nonlinear solver
- Concluding remarks

A wide variety of mathematical problem types depending on application

- **Steady-state simulation**
  - Sets of nonlinear algebraic equations
  - Sets of integral, partial differential and algebraic equations
- **Dynamic simulation**
  - Sets of differential & algebraic equations
  - Sets of integral, partial differential and algebraic equations
- **Plant design**
  - (mixed integer) nonlinear programming
  - (mixed integer) dynamic optimisaiton
- **Plant operation and control**
  - Linear programming
  - Nonlinear programming
  - Dynamic optimisation
- **Production planning & scheduling**
  - (mixed integer) linear programming

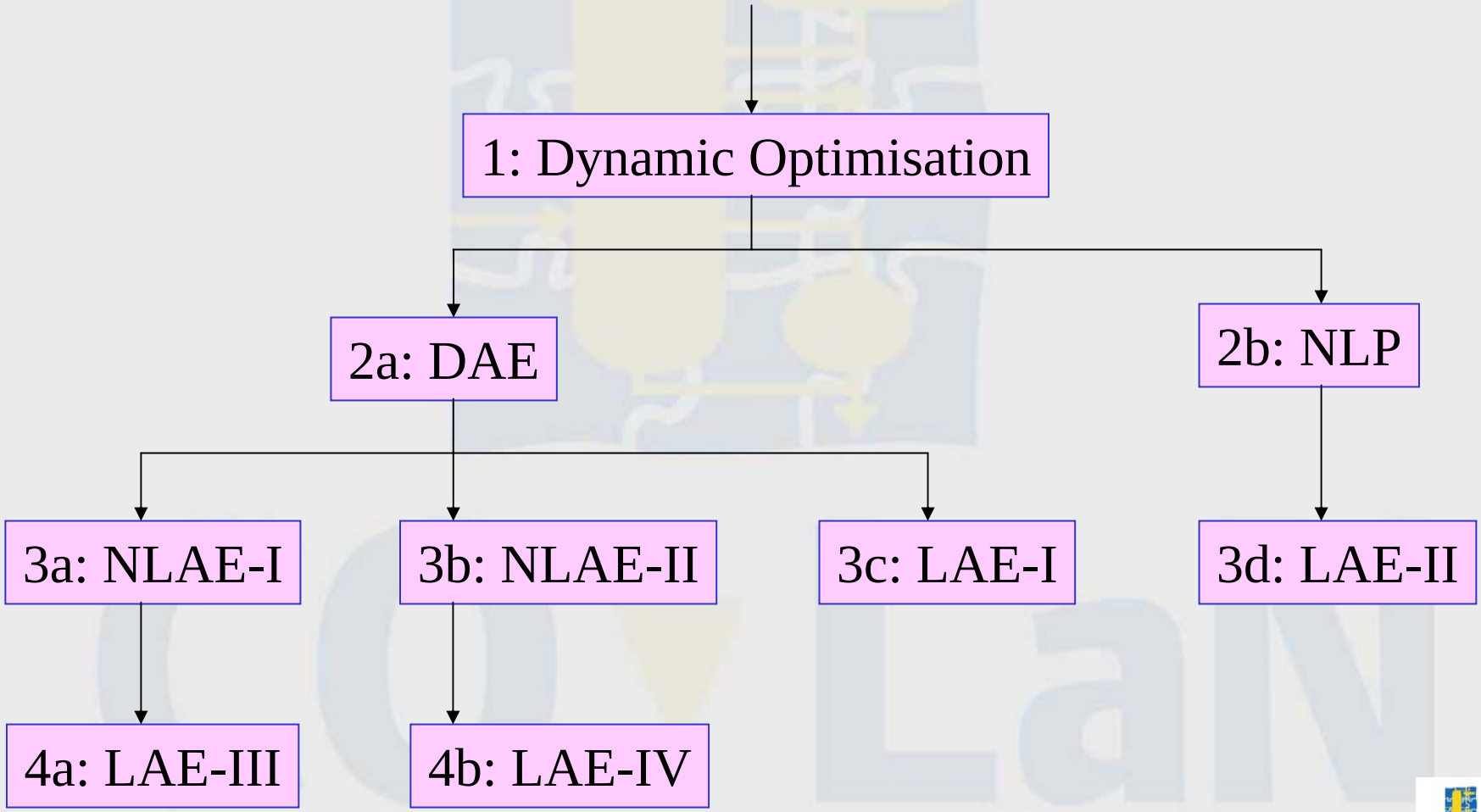
A hierarchy of solvers

- **To do steady-state simulation of a distillation column**
  - ➲ **we need to solve sets of nonlinear algebraic equations…**
    - **which involves solving linear equations…**

- **To optimise the grade transition in a polymerisation reactor**
  - ➲ **we need to solve a dynamic optimisation problem**
    - **which involves solving sets of DAEs**
      - **which involves solving linear algebraic equations**
      - **…and also solving nonlinear algebraic equations (2 different types)**
        - » **which involves solving linear equations**
    - **.. and also nonlinear programming problems**
      - **which involves solving (more) linear equations…**

Solver hierarchies

Activity: optimise the dynamic response of a unit



Overview

- Background and scope of CAPE-OPEN Numerical Solvers
- The challenge
- **General principles of CAPE-OPEN Numerics interfaces**
- Illustration – implementing simple nonlinear solver
- Concluding remarks

CAPE-OPEN objectives for numerical solvers

- **Allow the usage of solvers from different sources to perform all the activities supported by process modelling tools**
  - **complete "mix-and-match"**
  - **…at *any* point in the hierarchy**
  - **…not just at the top activity level**

- **Provide access to the mathematical statements of the models**
  - **allows the construction of third-party software to perform activities that are *not* directly supported by process modelling tools**
    - **e.g. model-based fault detection**

CAPE-OPEN Problem Objects

- **Fundamental principle: complete separation between**
  - **the description of the problem being solved**
  - **the code used for its solution**
- **Describe different types of mathematical problems as different software object classes with formally defined interfaces**

Mathematical Problem Type	CAPE-OPEN Problem Object
Nonlinear algebraic equations	Equation Set Object (ESO)
Differential-algebraic equations	Differential Algebraic ESO (DAESO)
Partial differential-algebraic equations	Partial Differential Algebraic ESO (PDAESO)
Mixed integer nonlinear programming problems	`minlp` object

CAPE-OPEN System Factories & Systems

- Classify numerical solvers into different categories
  - Nonlinear algebraic equation solvers
  - Differential-algebraic equation solvers
  - . . . . . .
- Each CO-compliant solver provides a Solver Manager
- A CO System can be constructed by applying a CO Solver Manager to a CO Problem Object

$$\left\{\begin{matrix}\text{CAPE}-\text{OPEN}\\ \text{System}\end{matrix}\right\} \equiv \left\{\begin{matrix}\text{Problem}\\ \text{being solved}\end{matrix}\right\} + \left\{\begin{matrix}\text{Numerical code}\\ \text{used for solution}\end{matrix}\right\}$$

CAPE-OPEN Systems

$$\left\{\begin{array}{c}\textbf{CAPE}-\textbf{OPEN}\\ \textbf{System}\end{array}\right\} \equiv \left\{\begin{array}{c}\textbf{Problem}\\ \textbf{being solved}\end{array}\right\} + \left\{\begin{array}{c}\textbf{Numerical code}\\ \textbf{used for solution}\end{array}\right\}$$

- Each CAPE-OPEN System has method(s) for solving the problem
- Final solution of the problem is placed in the CO Problem Object

Overview

- Background and scope of CAPE-OPEN Numerical Solvers
- The challenge
- General principles of CAPE-OPEN Numerics interfaces
- **Illustration – implementing simple nonlinear solver**
- Concluding remarks

CAPE-OPEN: how to solve a numerical problem

- **Example:**

  - ➲ **Do a steady-state simulation of a distillation column**

CAPE-OPEN: how to solve a numerical problem

- Step 1: Construct a CO Problem Object
  - e.g. an Equation Set Object containing the column equations, `columnESO`
- Step 2: Obtain a CO-compliant Solver Manager for an appropriate solver
  - e.g the BDNLSOL solver from PSE Ltd., `bdnlsolMGR`
- Step 3: Pass the Problem Object to the Solver Manager to obtain a CO System
  - e.g. `mySystem = bdnlsolMGR->CreateNLSystem(columnESO);`
- Step 4: Ask the System to solve itself
  - e.g. `mySystem->Solve();`
- Step 5: Retrieve the solution from the CO Problem Object
  - e.g. `X = columnESO->GetVariables() ;`

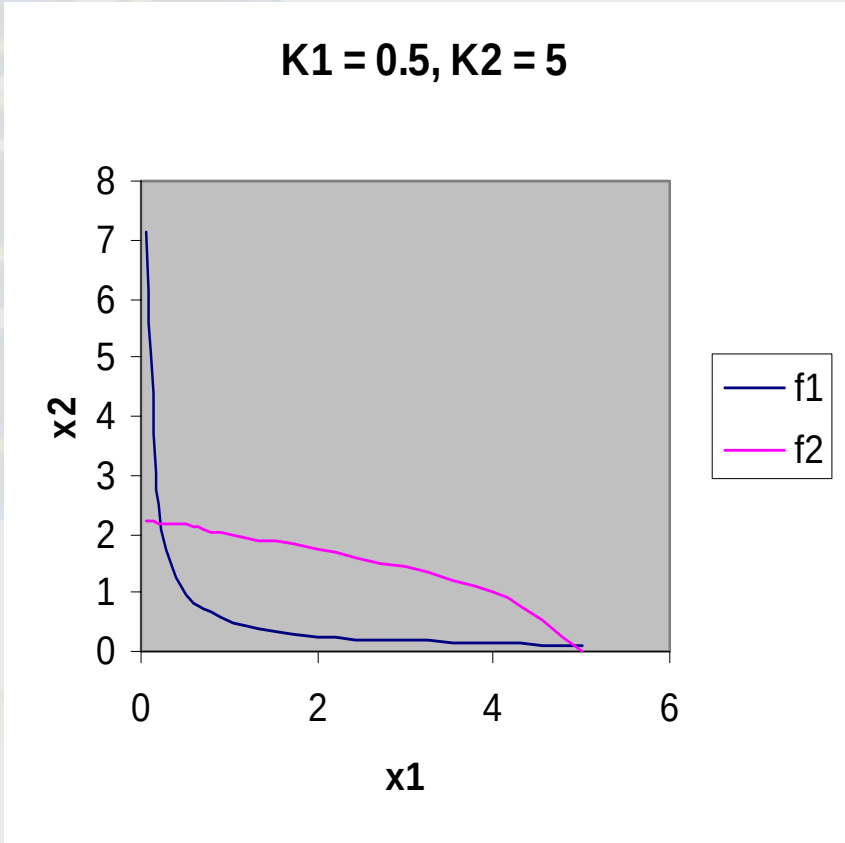
Under the hood – interaction between solver and ESO

- Excursion – nonlinear solver for specific problem
- Next stage of generalisation – hand-coded user functions
- Finally – the ESO: no coding!

Example problem

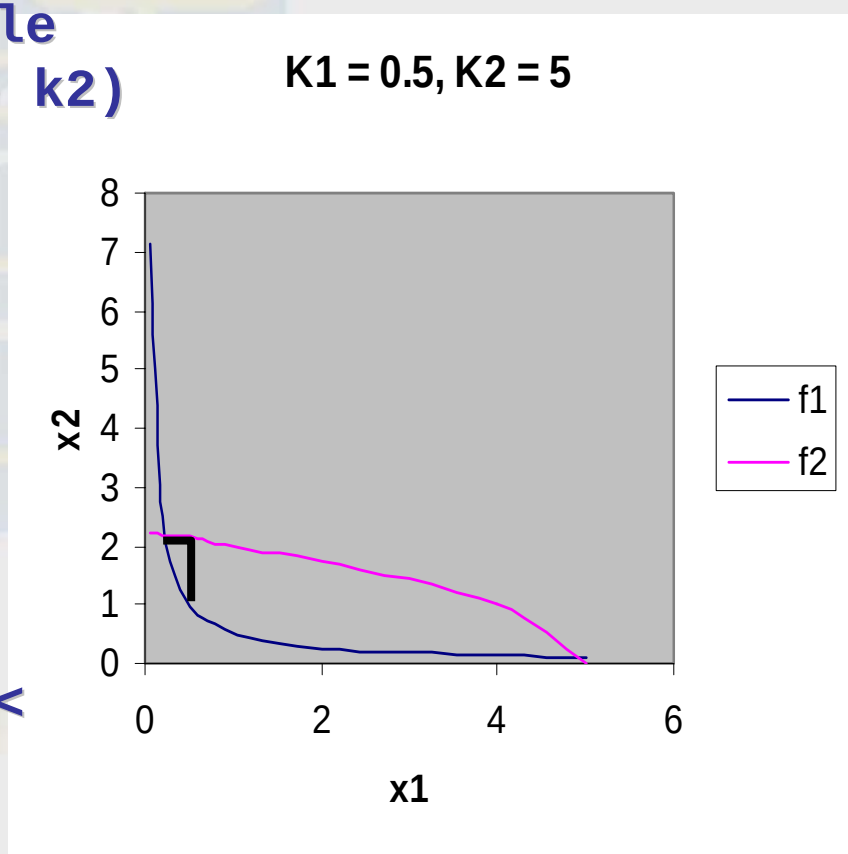
$$x_1 + x_2^2 = K_1$$

$$x_1 x_2 = K_2$$



Stage 1: specifically coded solution

```
void solve(double x1, double
   x2,    double k1, double k2)
// initial guess
x1 = sqrt(k2)
// Iteration
Int iter = 0
Repeat
    x2 = sqrt(k1 – x1)
    x1 = k2/x2
Until abs(k1 – x1 – x2^2) <
   eps
       OR iter = itermax
```



Stage 2: Hand-coded user functions…

$$x_1 + x_2^2 = K_1$$

$$x_1 x_2 = K_2$$

- **Residuals:**

```
void f(double x[], double k[], double r[])
    r[0] =  x[0] + x[1]^2 – k[0]
    r[1] = x[0] * x[1] – k[1]
```

- **Jacobian:**

```
void j(double x[], double k[], double
   A[][])
    A[0][0] = 1
    A[0][1] = 2 * x[1]
    A[1][0] = x[1]
    A[1][1] = x[0]
```

… but general solver

- Nlsolve(x[], double k[])

```
double A[neq][nvar]
double r[neq], delta[nvar]
niter = 0
Repeat
   call f(x, k, r)
   call j(x, k, A)

   call linsolve(A, r, delta)

   x = x – delta; niter++

Until abs(delta) < eps OR niter = itermax
```

Stage 3: Use the ESO

- **Residuals:**

```
void f(double x[], double k[], double r[])
   GlobalVarSet.SetAssignedVariables(k);
   UnknownsVarSet.SetAllVariables(x);
   ESO.GetAllResiduals(r);
```

- **Jacobian:**

```
void j(double x[], double k[], double A[][])
   GlobalVarSet.SetAssignedVariables(k);
   UnknownsVarSet.SetAllVariables(x);
   ESO.GetAllJacobianElements(A);
```

Note on ESO Implementation

- Basic idea is to treat equations (and derivatives) as data
- “Data” can now take the form of generated machine code, for speed
- Not necessary for solver to access symbolic form: only structure, residuals and Jacobian values

Stage 4: Componentise solver

- All solvers have this interface:

```
interface ICapeNumericSolverComponent :
   ICapeUtilityComponent {
     CapeSolverType GetType();
     void SelfDestruct();
     CapeArrayInterface GetParameters();
     CapeInterface GetParameterByName(in CapeString name);
     CapeArrayInterface GetStatistics();
     CapeInterface GetStatisticByName(in CapeString name);
};
```

- NLASystem is created using an ESO, and just adds Solve();

Stage 4: Evolution

- Nlsolve(x[], double k[])

```
double A[neq][nvar]
double r[neq], delta[nvar]
niter = 0
Repeat
   call f(x, k, r)
   call j(x, k, A)

   call linsolve(A, r, delta)

   x = x – delta; niter++

Until abs(delta) < eps OR niter = itermax
```

Stage 4: Evolution

- **SimpleSolver::Solve()**

```
CapeArrayDouble_var x
m_UnknownVarSet->GetAllVariables(x)
double A[neq][nvar]
double r[neq], delta[nvar]
niter = 0
Repeat
   call f(x, k, r)
   call j(x, k, A)

   call linsolve(A, r, delta)

   x = x – delta; niter++
   m_UnknownVarSet->SetAllVariables(x)
Until abs(delta) < eps OR niter = itermax
```

Stage 4: Evolution

- SimpleSolver::Solve()

```
CapeArrayDouble_var x
m_UnknownVarSet->GetAllVariables(x)
CapeArrayDouble_var A
CapeArrayDouble_var r, delta
niter = 0
Repeat
   call f(x, k, r)
   call j(x, k, A)

   call linsolve(A, r, delta)

   x = x – delta; niter++
   m_UnknownVarSet->SetAllVariables(x)
Until abs(delta) < eps OR niter = itermax
```

Stage 4: Evolution

```
SimpleSolver::Solve()
CapeArrayDouble_var x
m_UnknownVarSet->GetAllVariables(x)
CapeArrayDouble_var A
CapeArrayDouble_var r, delta
niter = 0
Repeat
   call m_ESO->GetResiduals(r)
   call j(x, k, A)

   call linsolve(A, r, delta)

   x = x – delta; niter++
   m_UnknownVarSet->SetAllVariables(x)
Until abs(delta) < eps OR niter = itermax
```

Stage 4: Evolution

```
SimpleSolver::Solve()
CapeArrayDouble_var x
m_UnknownVarSet->GetAllVariables(x)
CapeArrayDouble_var A
CapeArrayDouble_var r, delta
niter = 0
Repeat
   call m_ESO->GetResiduals(r)
   call m_ESO->GetJacobianValues(A)

   call linsolve(A, r, delta)

   x = x – delta; niter++
   m_UnknownVarSet->SetAllVariables(x)
Until abs(delta) < eps OR niter = itermax
```

Stage 4: Evolution

```
SimpleSolver::Solve()
CapeArrayDouble_var x
m_UnknownVarSet->GetAllVariables(x)
CapeArrayDouble_var A
CapeArrayDouble_var r, delta
niter = 0
Repeat
   call m_ESO->GetResiduals(r)
   call m_ESO->GetJacobianValues(A)
   call m_LinSol->SetMatrixValues(A)
   call m_LinSol->SetRHS(r)
   call m_LinSol->GetSolution(delta)
   x = x – delta; niter++
   m_UnknownVarSet->SetAllVariables(x)
Until abs(delta) < eps OR niter = itermax
```

Stage 4: Evolution

```
SimpleSolver::Solve()
CapeArrayDouble_var x
m_UnknownVarSet->GetAllVariables(x)
CapeArrayDouble_var A
CapeArrayDouble_var r, delta
niter = 0
Repeat
   call m_ESO->GetResiduals(r)
   call m_ESO->GetJacobianValues(A)
   call m_LinSol->SetMatrixValues(A)
   call m_LinSol->SetRHS(r)
   call m_LinSol->GetSolution(delta)
   x = x – delta; niter++
   m_UnknownVarSet->SetAllVariables(x)
Until abs(delta) < eps OR niter = itermax
```

Statistic: actual iterations

Parameter: Linear solver

Parameter: max iterations

Parameter: success condition

Stage 5: Package for gPROMS

- Create a DLL (Windows) or shared object (Unix) with a single exported function, “init” – say MySimpNL.dll
  - “init” function must return CORBA object reference to the solver’s SolverManager.
- Place DLL in gPROMS’ slv subdirectory.
- Tell gPROMS to use the solver:

```
SOLUTIONPARAMETERS
    NLSolver:=“MySimpNL”;
```

Note: for truly interoperable solvers, need to standardise these details!

Additional example: MINLP solver

gPROMS Solution parameters:

```
DOSolver := "CVP_SS" [
      "DASolver" :=  "DASOLV",
      "MINLPSolver" :=  "OAERAP" [
          "MaxIterations" :=  10000,
  "OptimisationTolerance" :=  0.0001,
             "MILPSolver" :=  "GLPK",
              "NLPSolver" :=  "SRQPD"
                              ]
                 ]
```

Additional e

- *Task:* Dynamic Optimisation
- *Interface:* IgDOSolverManager
- *Status:* Standardisation candidate

gPROMS Solver parameters:

```
DOSolver := "CVP_SS" [
      "DASolver" :=  "DASOLV",
      "MINLPSolver" :=  "OAERAP" [
          "MaxIterations" :=  10000,
  "OptimisationTolerance" :=  0.0001,
             "MILPSolver" :=  "GLPK",
              "NLPSolver" :=  "SRQPD"
                           ]
                ]
```

Additional example: MINLP solver

gPROM

- *Task:* Integration
- *Interface:* ICapeNumericDAESolverManager
- *Status:* Standard interface

```
DOSolver := "CVP_SS" [
      "DASolver" :=  "DASOLV",
      "MINLPSolver" :=  "OAERAP" [
          "MaxIterations" :=  10000,
  "OptimisationTolerance" :=  0.0001,
             "MILPSolver" :=  "GLPK",
              "NLPSolver" :=  "SRQPD"
                             ]
                 ]
```

Additional example: MINLP solver

gPROMS Solut

- *Task:* MINLP
- *Interface:* ICapeMINLPSolverManager
- *Status:* Standard interface

```
DOSolver := "CVP_SS" [
      "DASolver" :=  "DASOLV",
      "MINLPSolver" :=  "OAERAP" [
          "MaxIterations" :=  10000,
  "OptimisationTolerance" :=  0.0001,
             "MILPSolver" :=  "GLPK",
              "NLPSolver" :=  "SRQPD"
                             ]
                 ]
```

Additional example: MINLP solver

gPROMS Solution parameters:

```
DOSolver := "CVP_SS" [
      "DASolver" :=  "DASOLV",
      "MINLPSolver" :=  "OAERAP" [
          "MaxIterations" :=  10000,
  "OptimisationTolerance" :=  0.0001,
             "MILPSolver" :=  "GLPK",
              "NLPSolver" :=  "S    D"

                        ]
```

- *Task:* MILP
- *Interface:* ICapeMILPSolverManager
- *Status:* Standard interface

Additional example: MINLP solver

gPROMS Solution parameters:

```
DOSolver := "CVP_SS" [
      "DASolver" :=  "DASOLV",
      "MINLPSolver" :=  "OAERAP" [
          "MaxIterations" :=  10000,
  "OptimisationTolerance" :=  0.0001,
             "MILPSolver" :=  "GLPK",
              "NLPSolver" :=  "SRQPD"

                          ]
```

- *Task:* NLP
- *Interface:* ICapeMINLPSolverManager
- *Status:* Standard interface

Additional example: MINLP solver

gPROMS Solution parameters:

```
DOSolver := "CVP_SS" [
      "DASolver" :=  "DASOLV",
      "MINLPSolver" :=  "OAERAP" [
          "MaxIterations" :=  10000,
  "OptimisationTolerance" :=  0.0001,
             "MILPSolver" :=  "GLPK",
              "NLPSolver" :=  "IPOPT"

                          ]
```

- *Task:* NLP
- *Interface:* ICapeMINLPSolverManager
- *Status:* Standard interface

Additional example: MINLP solver

gPROMS Solution parameters:

```
DOSolver := "CVP_SS" [
        "OutputLevel" :=  0,
           "DASolver" :=  "DASOLV",
        "MINLPSo
             "Max
    "Optimisatio
                "Output
                 "MILPSolver" :=  "GL
                  "NLPSolver" :=  "IPOPT"
                                    ]
                        ]
```

- *Task:* NLP
- *Interface:* ICapeMINLPSolverManager
- *Status:* Standard interface

Concluding remarks

- CAPE-OPEN promotes standardisation of solvers for modular and equation-orientated modelling tools
- A wide variety of mathematical problem types to support a range of model-based activities
  - simulation, optimisation, parameter estimation
- Support for state-of-the-art in numerical solvers
  - interfaces make available all necessary information
- Enhance supply and usage of numerical solver technology from a variety of sources
  - universities & research institutes
  - specialist commercial providers
- Support development of more model-based applications